Deep Dive into Model-Driven Design

Website: https://www.avanscoperta.it/en/training/deep-dive-into-model-driven-design-workshop/

Trainer: Bruno Boucard and **Kenny Baas-Schwegler**

Supple Design

Supple design is a sophisticated approach to software development that focuses on creating a codebase that is a pleasure to work with, inviting to change, and deeply connected to the domain model. It's about crafting software that meets users' needs and facilitates developers in the continuous evolution of the application. Here's how you can strive for a supple design in your projects:

Embrace Deep Modeling

Start by thoroughly understanding and representing the domain within your code. This involves identifying and making explicit all the implicit concepts and relationships that define the domain. A deep model is a foundation for a supple design, enabling developers to make informed decisions that align with the domain's intrinsic characteristics.

Focus on Clarity and Simplicity

A supple design is characterized by its simplicity and clarity. This doesn't necessarily mean minimal code; instead, it's about ensuring that every piece of code has a clear purpose and is easy to understand and work with. Simplify interfaces, minimize dependencies, and ensure your codebase effectively communicates its intent.

Promote Iterative Refinement

Iterative refinement is central to achieving a supple design. This process involves continuously revisiting and revising the codebase to deepen the domain model, simplify complex areas, and eliminate unnecessary complications. Encourage regular refactoring sessions that focus on improving the code's structure and enhancing its alignment with the domain model.

Foster Collaboration with Domain Experts

Maintain a close collaboration with domain experts throughout the development process. Their insights can help identify areas of the model that need refinement and ensure that the software remains true to the domain's complexities and nuances. This ongoing dialogue is critical to evolving the domain model and, by extension, the software design.

Apply Design Patterns Thoughtfully

- **Intention-Revealing Interfaces**: Designing interfaces that communicate their purpose and usage without requiring deep knowledge of their implementation.
- **Side-Effect-Free Functions**: Creating functions that don't alter the system's state, making them predictable and easy to combine.
- **Assertions**: Using assertions to make the effects of operations explicit, aiding in understanding and reliability.

- **Conceptual Contours**: Identifying and aligning the model with the fundamental divisions in the domain enhances model coherence and simplifies design.
- **Standalone Classes**: Reducing dependencies to make classes self-contained and more accessible to comprehend.
- **Closure of Operations**: Designing operations that return a result of the same type as their input, simplifying interfaces, and reducing dependencies.

Declarative Design

We are moving towards designs that specify what the system should do rather than how to make software more accessible to reason about and more robust against changes. This includes using domain-specific languages to create highly expressive, domain-aligned code.

The essence of the text is that software design should not only meet user needs but also accommodate the developers who maintain and evolve it. By employing strategies that reduce complexity, enhance clarity, and embrace domain-driven principles, software can become more adaptable, easier to work with, and ultimately more effective in achieving its goals.

Prioritize Understandability and Predictability

Make your codebase understandable and predictable. This involves ensuring that developers can easily reason about the code's behavior, anticipate the effects of changes, and integrate new features without unintended consequences. Achieving this clarity and predictability makes the codebase easier to work with and adapt.

Conclusion

Achieving a supple design is a challenging but rewarding endeavor. It requires a deep commitment to understanding the domain, a willingness to engage in continuous learning and refinement, and a collaborative approach that leverages the collective expertise of the development team and domain experts. By striving for a supple design, you can create software that not only meets the needs of its users but also evolves gracefully in response to new insights and requirements.